NOAA Technical Memorandum NOS NGS 35

AUTOMATIC DETECTION OF LOOPS IN LEVELING NETWORKS

Rockville, Md.
July 1982

U.S. DEPARTMENT OF COMMERCE / National Oceanic and Atmospheric Administration / National Ocean Survey

NOAA Technical Memorandum NOS NGS 35

AUTOMATIC DETECTION OF LOOPS IN LEVELING NETWORKS

Edward H. Herbrechtsmeier

National Geodetic Survey
Rockville, Md.
July 1982

For sale by the National Geodetic Information Center,
Rockville, MD 20852 Price $2.40

UNITED STATES DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Oceanic and Atmospheric Administration
John V. Byrne, Administrator

National Ocean Survey
Herbert R. Lippold, Jr., Director

CONTENTS

Abstract.. 1

Introduction.. 1

The loop generation algorithm..................................... 2

Special cases... 5

Implementation and performance.................................... 6

Acknowledgment.. 7

Bibliography.. 7

Appendix. Program description and user's guide.................... 8

AUTOMATIC DETECTION OF LOOPS IN LEVELING NETWORKS

Edward H. Herbrechtsmeier
National Geodetic Survey
National Ocean Survey, NOAA
Rockville, Md. 20852

ABSTRACT. This publication describes an algorithm for finding a minimal set of fundamental loops in a network and discusses the application of the algorithm to leveling networks.

INTRODUCTION

Loops in leveling networks have historically been used for two purposes. First, the observed elevation differences around a loop are expected to sum to zero. Any misclosure (deviation of this sum from zero) is assumed to be due to observational, recording, or computing errors. The misclosure of a loop has been a primary tool for controlling the quality of field observations and computations, with various tolerances on the loop misclosure corresponding to various instrumentation and observing procedures. Misclosures in a network of loops may be used to isolate blunders. If two neighboring loops have large misclosures of nearly equal magnitude but opposite sign, then one or more blunders may exist in the common segment.

The second use for leveling loops has been to serve as a basis for adjustment by condition equations. The condition that the adjusted elevation differences around each loop must sum to zero gives rise to one condition equation for each loop. However, it may be shown that only $\ell = n - b + 1$ of these loops are independent, where b is the number of bench marks and n is the number of observations between bench marks. To use loop closures as a basis for adjustment, it is necessary to find a set of exactly independent loops. The common practice has been to use experience and intuition, as well as a sketch of the network, to find a satisfactory set of independent loops. To my knowledge, there has previously been no satisfactory automatic algorithm for finding such a set of loops. At least one algorithm for finding an independent set of loops in a horizontal network has been published (Steeves 1978), but it is not suitable for the task at hand because it does not find a minimal set of loops.

Most adjustments of leveling nets are now done by computer programs. Most of these programs, including the LEVEL1 program used by the National Geodetic Survey, are based on observation equations rather than condition equations. The problem of finding an independent set of loops for the purpose of adjustment has, therefore, largely disappeared, although the use of loops for isolating blunders in observations prior to adjustment remains an important problem.

The National Geodetic Survey has previously used network sketches to find the loops of a network. However, this method is time-consuming, labor-intensive, and generally creates a bottleneck in a system where most other tasks are performed on the computer. Furthermore, a sketch does not always adequately reflect the data at hand. The algorithm described here is designed to replace the process of determining loops from sketches. The design criterion for the algorithm is based on the following requirement. It should automatically "find that set of loops which would have been found by an analyst working with a sketch of the network." This turns out to be an impossible task, since the decisions made by the analyst with a sketch are not precisely defined, especially for a complicated network. Therefore, the criterion was modified to require agreement with the loops found by the analyst with a sketch "most of the time and especially for simple networks." This criterion was met.

THE LOOP GENERATION ALGORITHM

The objective of this algorithm is to generate a fundamental set of loops that are minimal or at least near minimal (defined below). The algorithm operates on the graph associated with the leveling network and assumes:

1. The length of each edge is known.
2. The graph is not disjoint.
3. The graph may or may not be planar.
4. The edges do not have directions associated with them.
5. The graph contains n edges and b vertices.

For a given graph, it may be possible to find many loops. Certain subsets of loops are said to be fundamental sets if all of the possible loops in the graph can be generated by combining (in an algebraic sense) loops from the subset. The number of loops, ℓ, in a fundamental set is given by

$$\ell = n - b + 1.$$

A fundamental set is said to be minimal if the sum of the lengths of the loops in the fundamental set is the smallest of all the possible fundamental sets. An individual loop is said to be minimal if it cannot be decomposed into a number of smaller loops.

The algorithm has three parts. In the first part, the graph is simplified and some or all of the loops may be found (i.e., the second and third parts may not be needed). The second part of the algorithm forms a spanning tree, having the property that the sum of the lengths of its edges is minimal, and generates an ordered list of edges that are not in the spanning tree. No loops are found in the second part of the algorithm. The third part of the algorithm uses the spanning tree and the list of edges to find the remaining loops. All of the loops found in the first part are minimal while some of the loops found in the third part may not be minimal. The first part of the algorithm is not a necessity, but its use makes the second and third steps computationally easier and more efficient.

Part 1

Second-degree vertices, spurs, and self-closing loops are removed from the graph of the network in this part of the algorithm. Second-degree vertices are removed because their absence greatly reduces the computational burden of the other portions of the algorithm. This is accomplished by replacing the series of edges by "equivalent" edges. For example, consider a set of n vertices v_1, v_2, ..., v_n. Assume that the degree of v_1 and v_n is not 2 and that the degree of v_2, v_3, ..., v_{n-1} is 2. This implies a set of edges e_1, e_2, ..., e_{n-1} connecting the vertices. These edges can be replaced by one equivalent edge, e', whose length is equal to the sum of the replaced edges. This also eliminates the vertices v_2, v_3, ..., v_{n-1}.

Spurs are defined as edges that are incident to vertices of degree 1, which implies that such edges cannot lie within a loop. Spurs are simply deleted from the graph. Note that when a spur is removed, the vertex of degree 1 to which it was incident is also removed.

Self-closing loops are defined as loops that pass through, at most, one vertex of degree 3 or higher. Hence, no portion of a self-closing loop can be a portion of any other loop. Therefore, self-closing loops are, by definition, minimal loops. Finding the self-closing loops here simplifies the logic in the third part of the algorithm. Once found, they are removed from the graph by deleting the edges in the loop. This also removes all second-degree vertices in the loop from the graph.

These three steps (remove second-degree vertices, remove spurs, find and remove self-closing loops) are repeated until the graph contains no second-degree vertices, spurs, or self-closing loops. The repetition is required because the removal of a spur can create a self-closing loop or a second-degree vertex, and the removal of a self-closing loop can create a spur or a second degree vertex.

Part 2

This part of the algorithm develops a spanning tree (breadth first) and an ordered set of "closing edges" for the graph generated in Part 1. A closing edge is defined as an edge that is not in the spanning tree and is incident to a different branch of the spanning tree at each of its endpoints. In fact, every edge that is not in the spanning tree is a closing edge.

The spanning tree will be generated such that the distance from the starting vertex, or root, to any other vertex will be minimal. This is done by choosing edges for the tree from a list of candidate edges, each of which has a score associated with it. The score for the candidate edge is computed as follows: An edge becomes a candidate when one of its endpoints is added to the spanning tree. This endpoint will be called the near end, i.e., the end nearest the root of the graph. The other endpoint will be called the far end. The score for the edge is then the distance in the tree from the root to the near end plus the length of the edge.

There are several preliminary steps to the algorithm. First, the starting vertex, or root, must be chosen. The choice of the root is arbitrary. The algorithm will need a table that gives the distance (in the tree) of all vertices in the tree from the root. This distance for the root is set to zero, and the distances for the other vertices are set to the -1 to indicate that they have not yet been added to the tree. The list of candidate edges initially contains all edges incident to the root. The list of closing edges is empty. The algorithm then proceeds as follows:

1. Select the edge with the lowest score from the candidate list, add it to the tree, and remove it from the candidate list.

2. Set the distance of the vertex at the far end of the edge added in Step 1 to the score of that edge. Call this vertex the new vertex.

3. Test each edge incident to the new vertex and not in the tree. If the edge is in the candidate list, move it to the list of closing edges; otherwise compute its score and move it to the candidate list. If several closing edges are found at the new vertex, they are put into the list of closing edges according to their score, i.e., lowest score first.

4. If the list of candidate edges is not empty, go to Step 1. When the list of candidate edges is empty, the spanning tree is complete and all closing edges have been found.

Part 3

This part of the algorithm uses the spanning tree and the list of closing edges generated in Part 2 to find the remaining loops. One loop is found for each edge in the list of closing edges. The spanning tree will have edges added to it in this part of the algorithm. Therefore, it will no longer be a spanning tree and will be referred to as the subgraph.

The following steps are performed for each edge in the list of closing edges, in the order of the list of closing edges:

1. Find the shortest path in the subgraph that connects the endpoints of the closing edge. This path and the closing edge form a loop.

2. If the path contains more than one edge, add the closing edge to the subgraph. If the path contains only one edge, the closing edge is parallel to the edge that forms the path and is, therefore, not added to the graph.

The algorithm used to compute the shortest path between the endpoints of the closing edge is similar to the algorithm used to build the spanning tree. A "shortest path tree" is started at one end of the closing edge. Each edge in the subgraph incident to this vertex is scored and put into a list of candidates. (The closing edge is not in the subgraph.) The algorithm then proceeds as follows:

1. Select the candidate with the lowest score and put it in the shortest path tree.

2. If the newly selected edge is incident to the closing edge, go to Step 4.

3. Test each edge in the subgraph incident to the far end of the newly added edge. If the edge has already been scored, remove it from the candidate list. (It cannot be part of the shortest path.) Otherwise, score it and add it to the candidate list. Go to Step 1.

4. The shortest path is found by starting at the end of the closing edge opposite the root of the shortest path tree and then by following the shortest path tree down to its roots.

SPECIAL CASES

In applying this algorithm to leveling networks, the real objective is to generate the loops that one "sees" as minimal in a network sketch. The algorithm will not always do this because the length of a path followed by a leveling crew does not represent the straight line distance between bench marks. For instance, consider the portion of a leveling network shown in figure 1.

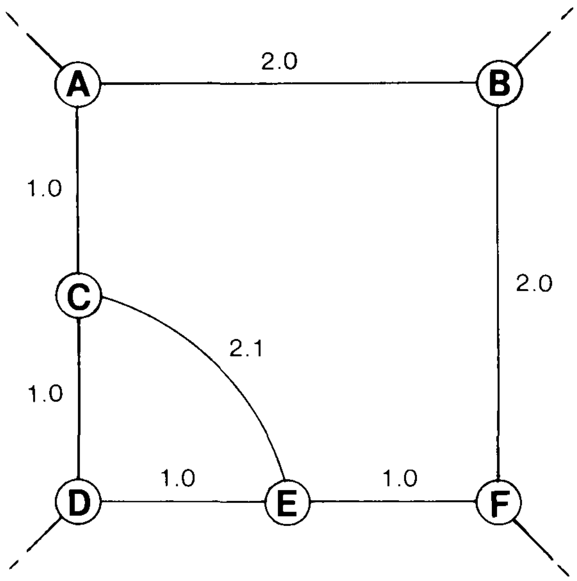


Figure 1.--A special case.

The numbers indicate the relative lengths of the links. The loop seen as minimal are ABFECA and EDCE. The algorithm will find EDCE correctly, but it will find ABFEDCA instead of ABFECA because the path EDC is shorter than EC. This situation can occur whenever the network contains a three-sided loop in which the sum of the lengths of two of the sides is shorter than the length of the third side.

The algorithm may appear to fail for networks that cannot be represented by a planar graph.

In figure 2, the algorithm will find the loops ABFEA, BCGFB, CDHGC, and AEHDA plus three of the following four loops: EFGE, EGHE, EFHE, and FGHF. This happens because the four "outer" loops, plus three of the "inner" loops, form a fundamental set. In other words, the closure of the fourth inner loop (i.e., the one not found by the algorithm) can be derived from the closures of the other three inner loops.

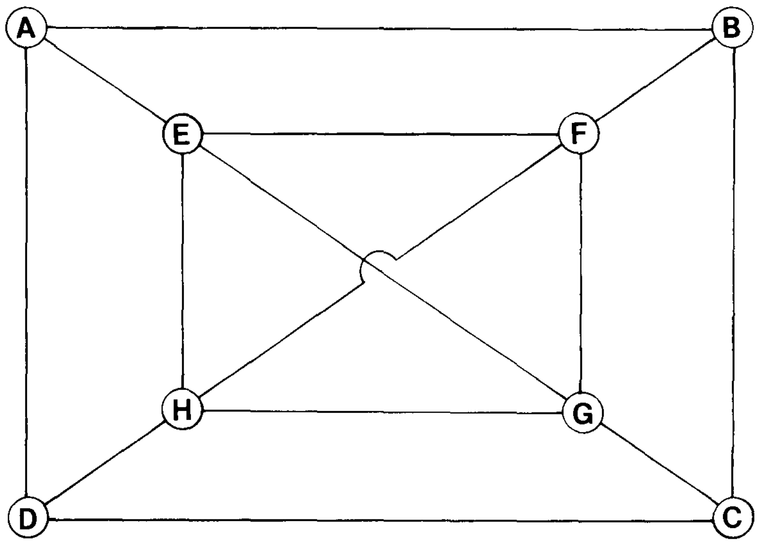


Figure 2.--A nonplanar network.

IMPLEMENTATION AND PERFORMANCE

The algorithm has been implemented as a computer program named LOOP1. The program is written in the PL/I language for the IBM PL/I Optimizing Compiler and runs under the IBM MVS operating system. It may be run on any machine supporting this compiler and operating system with sufficient memory. The memory required is approximately 160K bytes plus 30K bytes for each 1,000 bench marks in the network.

The LOOP1 program is integrated into the NGS system for the processing of leveling data. The primary input is a LEVEL1 input file created by the SELECT program, which is used to interface application programs to the permanent vertical data base files, known as Working File and Vertical Synoptic File. With this input, LOOP1 can automatically compute the closures of a minimal or near minimal set of fundamental loops in the network without any instructions from the user. Alternatively, the user may explicitly request LOOP1 to compute the closure(s) of any loop(s) in the network. In addition, if the output file from a LEVEL1 adjustment of the network is available, LOOP1 will compute the correction rates between junction points. (See appendix.)

Detailed instructions for using this program are given in the appendix. The timing information listed in table 1 was obtained from runs made on an IBM 3033.

Table 1.--LOOP1 central processing unit (CPU) usage

Bench marks/ observations input	Vertices/ edges in graph after reduction	Self-closing loops	Other loops	CPU time (sec)
2542/2572	20/31	3	12	2.98
2664/2838	118/206	20	89	6.35
2179/2480	255/428	9	174	24.53

ACKNOWLEDGMENT

The author would like to thank Dr. Richard A. Snay of the National Geodetic Survey for suggesting the idea of using a spanning tree and an ordered set of edges and for his helpful discussions.

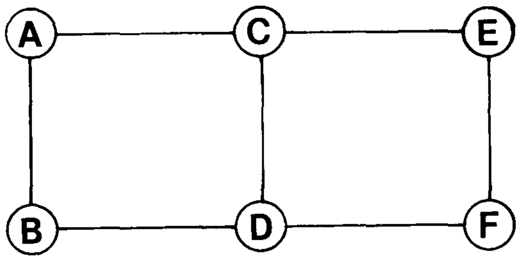
BIBLIOGRAPHY

Cribb, D. W., Ringelsen, R. D., and Shier, D. R., 1981: On cycle bases of a graph. Technical Report No. 362, Clemson University, Clemson, S.C.

Deo, Narsingh, 1974: Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall, Englewood Cliffs, N.J.

Steeves, Peter A., 1978: Economization of parametric horizontal control adjustment via condition equations. Proceedings of the Second International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Arlington, Va., April 24-28, pp. 535-554. National Geodetic Information Center, Rockville, MD 20852.

A fundamental set of loops is defined as a set whose members can be combined to yield the closure of any loop in the network. The number of loops in a fundamental set is equal to the number of links minus the number of junctions plus one. A minimal loop is a loop that cannot be decomposed into a set of shorter loops. Consider the following example:



All of the links in the above network are assumed to be of the same length. The network contains three loops: ACDBA, CEFDC, and ACEFDBA. The number of loops in a fundamental set is

$$\begin{aligned} L &= \text{number of links} - \text{number of junctions} + 1 \\ &= 7 - 6 + 1 = 2. \end{aligned}$$

Any two of the three loops in this network constitute a fundamental set. However, the minimal set of fundamental loops contains ABDCA and CDFEC. Note that these two loops can be combined to form the loop ABDFECA. Therefore, ABDFECA is not minimal because it can be decomposed into a set of shorter loops.

LOOP1 performs several transformations on a network to simplify computation. No changes are made to the input file. The first transformation that LOOP1 will make is the removal of multiple observations. Each set of multiple observations is removed and replaced by a single observation. The height difference for this new observation is the weighted mean of the height differences of the multiple observations, and its length and variance are equal to the respective mean values of the multiple observations. For each set of multiple observations, LOOP1 will list the new mean observation and each of the multiple observations along with their deviations from the mean height difference. Any observation that deviates from the mean by more than the allowable deviation will be flagged. This allowable deviation, A, is based on loop misclosure tolerances and is computed as

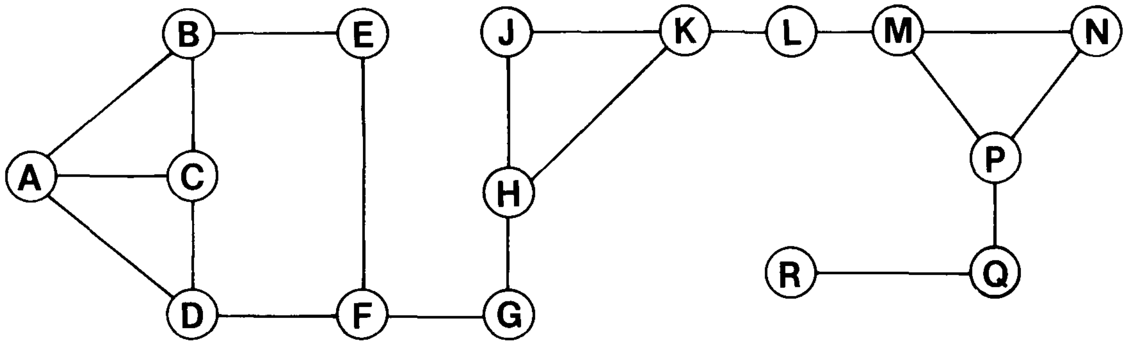
```
A = SQRT( 0.5 * MAX(0.250, k) * TOL**2 )
```

where

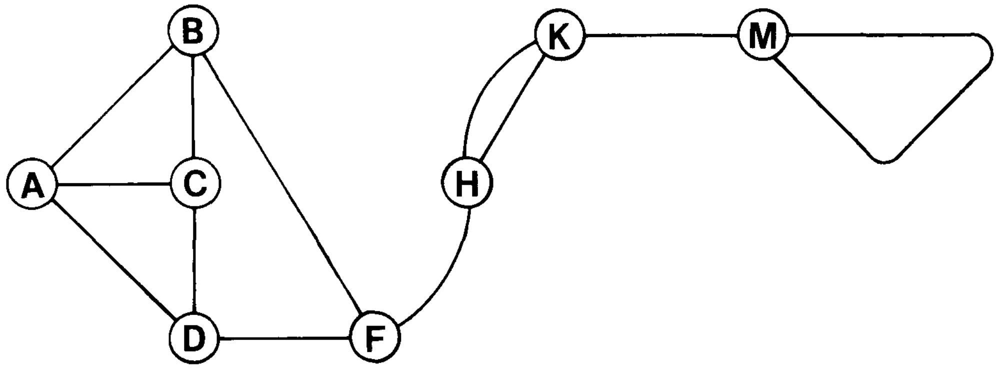
k = section length in kilometers,
TOL = loop misclosure tolerance in mm/SQRT(k) based on class and order.

If the section length is less than 0.25 km, then 0.25 km will be used for this computation.

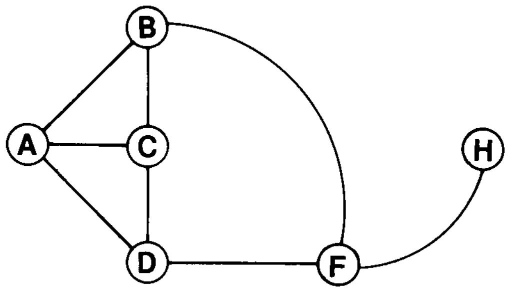
After LOOP1 has removed any multiple observations, it will remove any spurs, fold out second-degree marks, and remove any self-closing loops (spur loops). These three steps are repeated until all spurs, second-degree marks, or self-closing loops in the network are eliminated. These three steps must be repeated because the removal of a spur can generate a self-closing loop or a second-degree mark, and the removal of a self-closing loop can generate a spur or a second-degree mark. Consider the following example:



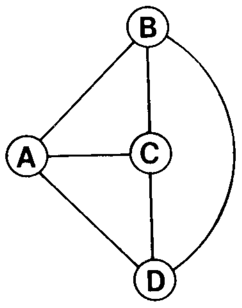
The spur observations PQ and QR will be removed. Note that this will make P a second-degree mark. The second-degree marks E, G, J, L, N, and P would be folded out next. The network would then look as follows:



As can be seen in the above diagram, a self-closing loop would exist at mark M. Upon removal of this self-closing loop the link KM would become a spur. This spur would then be removed which would cause mark K to become a second-degree mark. Mark K would be folded out and the resulting self-closing loop at mark H would be removed. The network would then look as follows:



The spur FH would then be removed. This would cause mark F to become a second-degree mark which would in turn be folded out. The network would then look as follows:



In this manner the original network containing 20 observations and 16 marks would be reduced to one network containing 6 links and 4 junctions. LOOP1 would then find the remaining loops using its auto-loop algorithm. It would list the closures for these loops and all of the self-closing loops.

The computation of the allowable misclosure for a loop is based on the order, class, and length of each section in the loop. An allowable variance for each section is computed as the tolerance squared times the section length. If the section length is less than 0.25 km, 0.25 km will be used for this computation. The tolerance is a function of the order and class of the survey. The allowable misclosure is computed as the square root of the sum of the allowable variances of the sections in the loop. LOOP1 will flag any loop whose misclosure exceeds the allowable misclosure.

LOOP1 can compute the correction rate for each link in the network if a LEVEL1 output file (i.e., adjusted elevation file) is available. The correction rate, C, is computed as

$$C = (DHA - DHO)/K$$

where

DHA = adjusted height difference in millimeters,
DHO = observed height difference in millimeters,

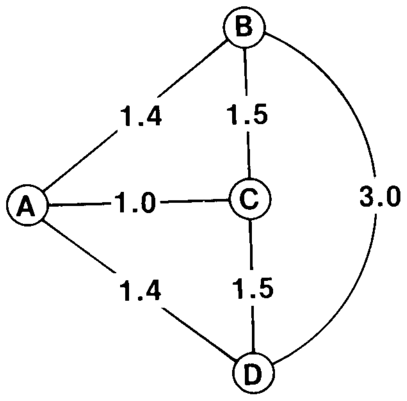
K = length of the link in kilometers.

This correction rate is the average rate for a link. Individual sections in a link may have different correction rates because of the weighting scheme employed in LEVEL1. Under this scheme, sections having the same length but different classes and/or orders will receive different weights in the adjustment.

LIMITATIONS

LOOP1 can accommodate networks containing up to 16,383 observations. This number is also the upper limit for programs SELECT and LEVEL1.

The objective of the automatic loop misclosure algorithm is to find the misclosures the user would "see" as minimal in a network sketch. However, LOOP1 cannot always do this. Consider the following example:



The numbers in the diagram indicate the relative length of the links. It would seem reasonable to expect that LOOP1 will find the loops ABCA, ACDA, and CBDC. However, instead of finding CBDC, it will find BDAB. This results from LOOP1 trying to find the shortest path between B and D that can be combined with link BD to form a loop. The path BAD has a length of 2.8 whereas the path BCD has a length of 3.0; thus LOOP1 will combine BAD with CD to form BDAB.

JOB SETUP

```
//X9Z JOB (5417,CB1,1,9),'name - bin#'
//PROCLIB DD DSN=CN5420.S7D.PROCLIB,DISP=SHR
//STEP1 EXEC LOOP1,INIT=X9Z,ACNT=5417,UNIT=3330,
//  VOL=NGS003,DSN='LEVEL.NET'
//GO.SYSIN DD *
TITLE  SAMPLE RUN
OPTIONS   TIMING
LOOP    AB0001   (15)   AB0003   (23)   AB0005
LOOPX   (67)     AB0001
LOOP    AB0005   (23)   (212)   (110)   (7)
```

EXEC CARD PARAMETERS

The parameters on the EXEC card are described next. They are shown as parameter = default followed by an explanation.

ACNT=5417

This parameter is optional. It indicates the account under which the adjustment input file is stored.

INIT=

This parameter is required. It indicates the initials under which the adjustment input file is stored.

UNIT=3330

This parameter is optional. It indicates the type of disk drive on which the adjustment input file is stored.

VOL=NGS003

This parameter is optional. It indicates the name of the disk pack on which the adjustment input file resides.

DSN=

This parameter is required. It supplies the last portion of the data set name of the adjustment input file. For this, the name of the adjustment input file would be CN5417.X9Z.ADJIN.LEVEL.NET.

These parameters may be in any order on the EXEC card. Note that the EXEC card in the above example could have been coded equivalently as

```
//STEP1 EXEC LOOP1,INIT=X9Z,DSN='LEVEL.NET'
```

If a LEVEL1 output file exists for a network, the correction rates between junctions can be obtained by executing LOOP1A. For example,

```
//STEP1 EXEC LOOP1A,INIT=X9Z,DSN='LEVEL.NET'
```

The adjusted height of each junction will also be listed.

USER INPUT - FILE SYSIN

LOOP1 reads only columns 1-71 (inclusive) of the cards input through this file to make it more convenient to submit runs via SUPERWYLBUR. All of the cards in this file are free format; i.e., there are no specific column locations for the items on the cards (except they must be in columns 1-71). The items or fields on each card are to be separated by one or more blanks; do not use commas or other characters to separate fields.

Three types of cards can be input through this file, namely, TITLE, OPTIONS, and LOOP cards. All of these cards are optional; however, the DD card for file SYSIN must always be included. The TITLE card is used to input a title that will be printed as part of the heading on each page of output. Code the word TITLE in the first field and code the title in the second field.

The OPTIONS card is used to modify the output of LOOP1. Code the word OPTIONS in the first field followed by one or more of the options listed below. Each option has two possible values. They are listed as default value/optional value.

AUTO/NOAUTO

This option indicates whether or not the user wants LOOP1 to generate a set of loop misclosures automatically.

SCLOOP/NOSCLOOP

This option indicates whether or not LOOP1 is to list the misclosures of self-closing loops.

LISTCOM/NOLISTCOM

This option indicates whether or not LOOP1 is to list combined observations. The observations will be combined in any case.

NOTIMING/TIMING

This option indicates whether or not LOOP1 is to print a breakdown of its CPU usage.

Any number of OPTIONS cards may be input. If any option is included more than once, the last value found will be used.

The LOOP and LOOPX cards are used to specify additional loops for which the user wants misclosures. Listed next are some rules for specifying loops.

1. The first element in the specification must be an ACRN. (An ACRN is a code assigned by NGS to uniquely identify bench marks.)

2. If the last element in a specification is an ACRN, it must be identical to the first ACRN.

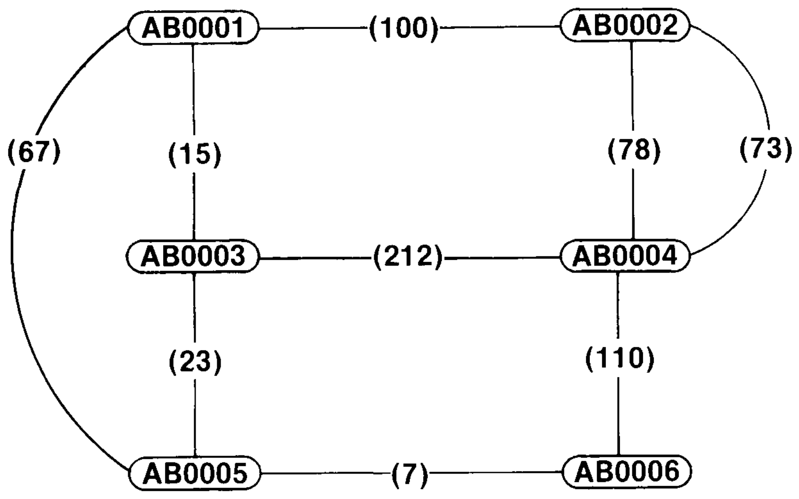
3. The path specified must be continuous.

4. The path specified must close on itself at the starting ACRN and only at the starting ACRN.

5. The path specified must not be ambiguous.

6. Link numbers must be enclosed in parentheses. Blanks are not allowed inside parentheses.

There are several ways to specify the path of a loop. These will be explained with the aid of an example.



The numbers in parentheses in the network shown above are link numbers generated by LOOP1. To request the computation of the misclosure of a loop code a card with the word "LOOP" in the first field and code the ACRN of the mark at which the loop starts and ends in the second field. Code the link numbers and/or ACRNs that form the rest of the loop specification in the third and remaining fields.

Suppose that one wants LOOP1 to compute the closure of the loop AB0001 - AB0003 - AB0005 - AB0001. The most detailed specification would be

```
LOOP   AB0001  (15)  AB0003  (23)  AB0005
LOOPX  (67)  AB0001
```

Note the LOOPX card. It is used as a continuation of the preceding LOOP card. A LOOP card may be continued onto any number of LOOPX cards. The two cards cited could be read as follows: Start at mark AB0001 and proceed via link 15 to mark AB0003, then proceed via link 23 to mark AB0005, then proceed via link 67 to mark AB0001. As stated previously, this specification is the most detailed. Here are some alternative specifications that are equivalent:

```
LOOP  AB0001  (15)  (23)  (67)
LOOP  AB0001  AB0003  AB0005  AB0001
LOOP  AB0001  (15)  AB0005  (67)
```

The above example means that the first card starts at AB0001, then proceeds via links 15, 23, and 67, returning to AB0001. The second card starts at AB0001, then goes to AB0003, AB0005, and returns to AB0001. The third card starts at AB0001, proceeds via link 15 to AB0005, and returns via link 67 to AB0001.

In the rules for specifying loops, it was stated that the specification of the path of the loop must not be ambiguous. Such a situation can occur when more than one link exists between two junctions. For example, consider a loop that starts at AB0001, then goes to AB0002, then goes to AB0004, then goes to AB0003, and then returns to AB0001. There are two links between AB0002 and AB0004. In such a case, the user must specify which link is to be used by including the link number (either 73 or 78 in this case) of that link.

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Ocean Survey
National Geodetic Survey, C18x2
Rockville, Maryland 20852

OFFICIAL BUSINESS

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-210
THIRD CLASS MAIL

U.S.MAIL ®